Modeling temporal structure with LSTM for online action detection

Roeland De Geest
KU Leuven, ESAT-PSI, imec
roeland.degeest@esat.kuleuven.be

Tinne Tuytelaars
KU Leuven, ESAT-PSI, imec
tinne.tuytelaars@esat.kuleuven.be

Abstract

Online action detection is a challenging problem: a system needs to decide what action is happening at the current frame, based on previous frames only. Fortunately in real-life, human actions are not independent from one another: there are strong (long-term) dependencies between them. An online action detection method should be able to capture these dependencies, to enable a more accurate early detection.

At first sight, an LSTM seems very suitable for this problem. It is able to model both short-term and long-term patterns. It takes its input one frame at the time, updates its internal state and gives as output the current class probabilities. In practice, however, the detection results obtained with LSTMs are still quite low. In this work, we start from the hypothesis that it may be too difficult for an LSTM to learn both the interpretation of the input and the temporal patterns at the same time. We propose a two-stream feedback network, where one stream processes the input and the other models the temporal relations. We show improved detection accuracy on an artificial toy dataset and on the Breakfast Dataset [21] and the TVSeries Dataset [7], real-life datasets with inherent temporal dependencies between the actions.

1. Introduction

Action recognition is (and has always been) one of the main topics of interest in video interpretation. In early research, one often assumed that a segmented action video was given as input, and the action class needed to be determined out of a predefined list of actions (*e.g.*, [9, 23, 35]). Later, action detection (*e.g.*, [12, 18, 36]) drew more interest: given an unsegmented video, what action(s) occur in this video, and when do they start and end? Only recently, the problem of online action detection has started getting attention (*e.g.*, [7, 26]). In this setting, actions need to be found in a continuous stream of video data. The decision should be made as early as possible, ideally immediately after seeing a frame: what action, if any, is happening anywhere in this frame? Even though processing the video ideally happens in real time, online action detection is a different problem than the older problem of real-time action detection (*e.g.*, [3, 20]). Information from future frames is not available when a decision has to be made. This problem is more difficult, but has many practical applications, like surveillance and human-robot interaction, where anticipation and early reactions are crucial.

When dealing with continuous action streams, the long-term relations between actions become more important. Indeed, when an action has just started, the amount of direct evidence may be very limited. Luckily, many actions do not occur in a random order: they depend on each other. In many practical applications, they occur in a semi-predictable order – although there are still variations. Elderly people, for instance, go through a similar routine every day: they get out of bed, get dressed, prepare breakfast and read a newspaper. Even young people tend to adhere to fixed behavioral patterns and routines, *e.g.*, the order in which one makes a sandwich or salad. Another example are dancing videos, with dance moves occurring in typical sequences. In all these applications, the accuracy of the system could increase if it would learn to exploit the long-term relations between the actions or steps. In the task of multimedia event detection, videos containing certain events need to be retrieved from a large database. Since an event is composed of many different (shorter) actions, these methods often try to identify these actions and their co-occurrence or the order between them [6, 10, 34]. The videos contain only one event, however. We consider the more general case, where there can be dependencies between actions that do not belong to an easily defined higher-level event.

Early action recognition methods used hand-crafted features, either inspired by the features used for object recognition in still images (*e.g.*, [23]), or based on trajectories (*e.g.*, [35]). With the renewed interest (and excellent results) of neural networks on object recognition, there have been many proposals to use these networks for action recognition as well. Two main strategies have emerged. In the first one, a CNN is trained with multiple frames as input

(*e.g.*, [28]). Often, these CNNs are hierarchical to capture information from a large part of the video. The other strategy uses recurrent neural networks, often LSTMs (*e.g.*, [8]). Here, a network processes frame by frame and captures the information of previous frames in its internal state representation. Recurrent networks have been very successful for speech recognition and language generation. For action recognition (and especially action detection), however, they have not yet been proven to be clearly superior.

At first sight, LSTMs seem to be tailored for online action detection. They inherently take a framewise input and give an output every frame, with minimal delay. This way, a sliding window (with parameters that need to be tuned) is avoided and the computational efficiency is higher. They have the capability to model both short-term and long-term temporal dynamics. Yet, as shown in [7], simply applying an LSTM for online action detection does not result in significant improvements over more traditional methods.

In this work, we argue that online action detection, when both short-term and long-term dependencies are present, is too difficult to be solved by a simple recurrent neural network. Yet, recognizing these dependencies is important: at the start of an action, not enough information on the appearance is available, so a system should rely on the previous actions and extrapolate from there. We propose some changes to the model that can improve results.

In theory, an LSTM is very promising. In its most basic form, a representation for every frame is fed into the LSTM and it outputs the probabilities of action classes occurring in that frame. The representation is in most cases the output of a fully connected layer of a still image CNN, finetuned on the action dataset. This complicates the task of the LSTM: it not only needs to capture the temporal structure of the actions, but it also needs to learn a way to interpret the output of the fully connected layer of the CNN. The joint learning of these two completely different aspects seems to be a bit too much. From our first experiments (described in section 3.3), we learned that the LSTM can get stuck in a local minimum. It interprets the input quite accurately, yet the temporal dependencies of the actions often seem to be overlooked. The system outputs high probabilities for classes that, according to the training data, should never follow the previous classes.

Therefore, we design a new architecture that splits the subtask of the interpretation of the CNN features and the modeling of the temporal dependencies. Our two-stream feedback network uses two LSTM streams: one focuses more on representation, while the other is dedicated to the temporal structure. This approach improves the detection scores, both on a synthetic dataset (based on MINST [25]) and two real dataset with temporal dependencies between the actions: the Breakfast Dataset [21] and the TVSeries Dataset [7].

It should be noted that pose features are often used as input for the LSTM as well, instead of a fully connected layer. They are more easy to interpret, due to their low-dimensionality. However, they do not take object or scene cues into account, so this information is then permanently lost. We show that, even on a dataset with characteristic poses, the detection accuracy is a lot lower when using poses than when using the CNN features as input for the LSTM. Our new architecture improves the detection accuracy not only when using CNN features, but also when using pose features.

In the next section, we discuss the related work on online action detection. We present the standard LSTM, our synthetic toy dataset and our first experiment in section 3. In section 4, we explain our proposed strategy. Afterwards, we show experimental results in section 5. Section 6 concludes the paper.

2. Related work

Online action detection Hoai and De la Torre [15] are the first to present the problem of online action detection. They simulate the sequential arrival of training data and train a structured output SVM that enforces the score of frame $t+1$ to be higher than the score of frame t. They make some assumptions, however, that greatly simplify the problem. At test time, they know that there is only one action instance per video. When watching the video, they start detecting this action when a threshold of the scores is exceeded. Only at the end of the video, they decide the exact start and end time of this action. In their later work [16], they discuss a setting where multiple actions can occur per video, but they never evaluate this. In the follow-up work by Huang *et al.* [17], the problem is treated more like classification instead of detection. All possible classes are gradually eliminated until only one action remains, and this action is given as output. Their negative data is unrealistically simple: a person is just standing. Soomro *et al.* [31] extract a pose in every frame; they use these poses for a sliding-window approach with class-dependent window size. Here too, a structured output SVM makes sure the detection scores increase over time.

Other approaches use a random forest for online action detection. Baek *et al.* [1] implemented a skeleton-based method. During training, however, their forest has extra information available: CNN features of the original RGB and depth images, and temporal context that indicates the action progress. Garcia-Hernando and Kim [14] start from a skeleton representation as well and learn a random forest in such a way that the node criterion can depend on the information of the previous frames. The low-dimensionality of pose features makes them compatible with a random forest, however, a lot of possibly useful object and scene information is permanently lost.

Singh *et al*. [30] recently developed a method that generates candidate action bounding boxes in frames based on appearance and flow. These bounding boxes are incrementally grouped in action tubes, and those (partial) tubes get a class probability with Viterbi's algorithm. The tubes are classified independently, however, while we argue that it is important to capture the relations between actions as well.

As mentioned in the introduction, LSTM seems well-suited for the problem of online action detection, and a few methods take this approach. Becattini *et al*. [2] try to answer three questions: when is an action taking place, where is it taking place in the frame, and how far has it progressed yet? They use an R-CNN for the first two questions, and an LSTM for the last. They use CNN features as input for their system. Their detection network does not make use of temporal information, however, which seems a crucial factor to make an early decision: in the first frames of an action, the only discriminative information available is the occurrence of the previous frames. Li *et al*. [26] designed an LSTM for online action detection that gives scores for every class at every frame, and have a second output that estimates the start and end frame of the current action. They start from 3D skeletons. Our proposed method could easily be combined with theirs, however, they assume that actions occur independent from each other, in random order, as opposed to the main idea of our paper.

On realistic (TV series) data with a lot of background frames, De Geest *et al*. [7] have shown that a simple LSTM approach is not sufficient for online action detection, and even worse than the traditional pipeline of improved trajectories, Fisher vectors and SVM. Therefore, in this work, we try to boost the LSTM performance by helping it to capture the long-term dependencies between actions.

Datasets Several datasets have been used to test online action detection methods. Some of them, like UCF101 [32] and J-HMDB [19], are not suited for the problem, since the videos only contain one action class. Others, like MSRII [4] and the Online Action Detection Dataset [26] are collected in such a way that the different actions of a video occur independent from each other. As argued earlier, this is an invalid assumption in real life. The 50 Salads dataset [33] clearly has strong dependencies between the actions, but it is quite small with only a few hours of data. We will experiment on the TVSeries Dataset [7]. It contains realistic videos, but most annotated actions are not strongly dependent on each other. If more classes would have been annotated, this dataset would have been more suitable for our experiments.

We also test on the Breakfast dataset [21], a cooking dataset containing more than 70 hours of video. 52 persons make ten different breakfast-related recipes, in 18 different kitchens, and they are recorded from multiple viewpoints. 47 subactions are annotated. Strong temporal relations exist between these subactions: one has to crack an egg before frying the pancake. It can depend on the person, however, whether they pour milk first, or first crack the eggs. This makes the dataset very realistic, in line with our expectations. Until now, the authors of the dataset are the only ones that have experimented on it, and only for offline action detection. In their first work, Kuehne *et al*. [21] transfer techniques from speech recognition to action detection. They model every subaction with a separate hidden Markov model, based on even finer annotations. The ten recipes are then modeled as sequences of action units using a context-free grammar. As low-level input, they experiment with HOGHOF [24] and Trajecton [27] features per frame. These are grouped in a sliding window bag-of-words, and then fed to the HMM. In their follow-up work, Kuehne *et al*. [22] start from improved dense trajectories [35]. These are grouped with a sliding window and coded as Fisher vectors, and then reduced with PCA, because a HMM cannot cope with large input dimensions. The rest of the pipeline is similar to [21]. These methods are not readily compatible with online action detection: they assume a strict sequential progression of actions. A misdetection would be unrepairable.

3. An initial exploration

In this section, we describe the experiment that suggested the LSTM does not capture all temporal dependencies. The standard LSTM is described in section 3.1. The Noisy MNIST data we developed and used to quickly test strategies is introduced in section 3.2. The first experiment itself is described in section 3.3.

3.1. Standard LSTM

A long-short term memory (LSTM) network is composed of blocks with four main components: a memory unit c, an input gate i, a forget gate f and an output gate o. The network is modeled with the following equations, as a function of the time t. Here, σ is the sigmoid function.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

In action recognition, the input x_t is often a pose representation or the output of a fully-connected layer of a CNN trained for framewise action recognition. The output h of the LSTM layer is calculated as

$$h_t = o_t \tanh(c_t)$$

Sometimes, multiple layers are chained after each other: the x_t of a layer is then the h_t of the previous layer. An

Figure 1. A few example frames of our noisy MNIST dataset. The top row contains the same number '8' with different random noise, the bottom row is a sequence of '4'.

LSTM is trained by unrolling the network over time: the network is cloned and chained. Gradients can then back-propagate over time to update the trainable parameters W and b.

3.2. Noisy MNIST data

We created an artificial dataset to test our different strategies and parameters in a quick and easy way. Our dataset needed to have clear and strong temporal relations between the different classes to most closely simulate the temporal behavior of many practical real-life applications. We start from the MNIST dataset [25], which consists of handwritten numbers. The task here is to recognize the numbers. We create video clips from this dataset in the following way. We take a number and obtain a small video snippet by repeating this number for a certain amount of frames. The amount of repetition N depends on the number in the frames, but can vary a bit (randomly chosen in interval $[number + 1, number + 5]$). Gaussian white noise is applied to all frames with mean 0 and a variance that gradually decreases from 0.95 at frame 1 to 0.05 at frame N/2 and then again increases to 0.95 at frame N. Some frames are completely illegible by humans, others are a bit easier, as can be seen in figure 1. We design a transition matrix that defines the order of numbers. We use it to generate longer video sequences by concatenating the snippets. The MNIST training data is used to create training videos, the MNIST test data is used for test videos. In our experiments, we test on two different versions of this noisy MNIST dataset, with different transition matrices. In the MNIST-Easy dataset, every number can be succeeded by one of two other numbers. In the MNIST-Hard dataset, the transition matrix is generated randomly. Here, every sequence of numbers is possible, but the probability varies. Both datasets are balanced: every number occurs approximately an equal amount of times.

We trained the model developed by LeCun *et al*. [25] on the clean MNIST data (with static images). We evaluate it on our Noisy MNIST data and extract the fc2 features of this network. These features are taken as input for the LSTM in our experiments.

This dataset has strong temporal relations, both short-term (a number is repeated multiple times) and long-term (due to the transition matrix). It therefore resembles real-life, where a (sub)action takes a certain amount of time, and multiple (sub)actions form part of a larger action. We used this data for testing our strategies and optimization of the parameters. Even though this toy dataset does not contain actions, the input to the LSTM is similar for action detection. In both cases, the LSTM receives a feature vector that has already made abstraction of the real content of the frame. The exact feature vector depends not only on the content of the frame, but also on the noise in the frame and the way the number or action is actually shown. Due to our gradual decrease and increase of noise, the middle frame of a number sequence is most easily identified. This is similar to actions, where the middle frames are often most characteristic for the action. The Noisy MNIST dataset can quickly indicate which strategies are likely more successful on real action data. This dataset does of course not replace real experiments for online action detection on large datasets.

3.3. Motivational experiment

As mentioned before: an LSTM has difficulties with modeling both the interpretation of the input vector and the temporal dependencies at the same time. To demonstrate this, we conducted a simple experiment. We created a very easy MNIST video: it contains clean (non-noisy) sequences of numbers from 0 to 9. Every number is shown for 25 successive frames before it changes to the next number. This means that the upcoming frames can be predicted perfectly purely based on the temporal model. The CNN developed by LeCun *et al*. [25] achieves an accuracy of 98.5% on this data. An LSTM with the fc2 layer of this network as input can improve the accuracy to 99.9%. However, if we add noise to the frames (as described in section 3.2), the CNN score decreases to 61.9% (since it was not trained on noisy data). The LSTM score decreases as well, to 97.4%. Nevertheless, the temporal structure is exactly the same. If the LSTM would optimally make use of it, a score very close to 100% should be achievable.

4. Two-stream feedback network

Our experiment of the previous section showed that an LSTM not always captures all temporal dependencies in the dataset. In this section, we introduce a new architecture that mitigates this problem.

Instead of a (set of) standard LSTM layer(s), we designed a two-stream network. This network is specifically designed to have an upper stream that focuses on the interpretation of the frames, and a lower (temporal) stream

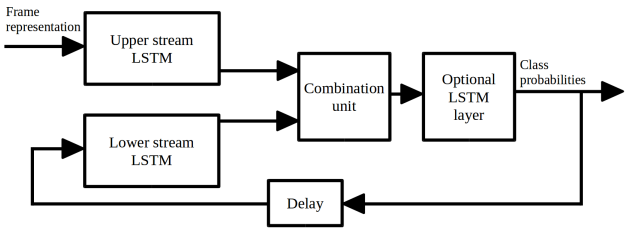

Figure 2. Basic structure of our two-stream architecture with feedback. The upper stream receives a frame representation (*e.g.*, CNN or pose features) as input. It learns to interpret this input and prepares an output for the combination unit, that combines the two streams. This unit can be followed by another LSTM layer, or generate class probabilities directly. The class probabilities of the previous frame are fed to the lower stream as input. This lower stream therefore never has information on the frame that is currently considered and must make its decision based on a good temporal model.

that captures the temporal dependencies. A combination unit merges the output of both frames, projects this representation to the number of classes and makes a final class prediction.

The upper stream is modeled as an LSTM layer. It receives as input a high-dimensional CNN representation of the frame. It should learn to map this representation to a lower-dimensional inner state. A fully-connected layer could in theory also reduce the dimensionality, but our experiments on Noisy MNIST data showed that better performance is obtained with an LSTM. Since this unit can also learn some (short-term) temporal relations, noise in the input does not necessarily propagate to the output of the upper stream.

The lower stream needs to capture the short-term and long-term temporal dependencies in the data. It is therefore modeled as an LSTM. We experiment with two strategies to force it to learn these dependencies. First, we show it the same representation as the first stream during training, with one exception: the last representation of each batch is put to zero. This way, the LSTM has no information on the content of the last frame. It can only make a prediction of the classes in that frame when it has a good temporal model. However, our experiments on Noisy MNIST data showed that this strategy does not help the detection accuracy. Since the inputs of the temporal stream are standard CNN features, the temporal stream already needs to have a good idea of the interpretation of the input. Our second strategy solves this problem by introducing a feedback loop (see figure 2). The temporal stream now receives the class probabilities of the previous frame as input. This input is low-dimensional, and interpreting it is a trivial task. This stream still has no information on the current frame, so it needs a good temporal model to create a reliable output.

We try multiple strategies for combining the output of the two streams: a summation, a concatenation or a weighted sum with one weight per stream. These weights are learned: a fully-connected layer takes the output of both streams as input and outputs the two weights. After the combination, the representation is projected to the number of classes with a fully-connected layer.

Formally, our network can be described as follows. The combination unit is defined as

$$y_t = h_{1,t} + h_{2,t}$$

for the summation (where $h_{1,t}$ is the output of the first stream LSTM and $h_{2,t}$ is the output of the second stream LSTM),

$$y_t =< h_{1,t}, h_{2,t} >$$

for the concatenation,

$$y_t = w_1 h_{1,t} + w_2 h_{2,t}$$

for the weighted sum, where

$$w_1 = \sigma(W_{w1} < h_{1,t}, h_{2,t} > + b_{w1})$$
$$w_2 = \sigma(W_{w2} < h_{1,t}, h_{2,t} > + b_{w2})$$

A fully-connected layer then maps this representation y_t to the final output z_t of our two-stream feedback network:

$$z_t = W_y y_t + b_y$$

The input of the temporal stream (as provided by the delay module) is then defined as z_{t-1}.

We optionally insert an extra LSTM layer between the combination and the fully-connected layer. The input of this LSTM layer is then y_t, the input of the fully-connected layer is the output h_t of the LSTM. This layer learns how to combine the information of the two streams, and can learn to detect bursts of unreliable output of one of the streams.

With this model, the combination unit is free to ignore one of the streams. If it learns to ignore the lower stream, the architecture would be identical for the two-stream network and the baseline (2-layer) LSTM. This way, the two-stream network can also be applied on data that does not contain strong dependencies between actions. It can then learn to only use the upper stream, provided enough training data is available.

Our model can be treated as a black box and replace the LSTM in other LSTM-based strategies to help them modeling the long-term dependencies.

5. Experiments

5.1. Dataset

5.1.1 Breakfast Dataset

The Breakfast Dataset [21] consists of 10 action classes related to breakfast preparation (like making pancakes or frying eggs). The actions are performed by 52 individuals in

18 different kitchens. Every instance is recorded from multiple viewpoints. In total, the dataset has 77 hours of video at a framerate of 15 fps. There are four splits for evaluation; we present results on the first split. Every video has annotations for 47 finer classes as well (like pouring milk or breaking egg). These subactions occur in a semi-predictable way: you first have to break an egg before you can mix it. It depends on the person, however, whether the milk or the eggs are added first when making pancakes. These are the long-term dependencies between the subactions. The short-term dependencies are between the frames of the same subaction.

This dataset has not yet been used for online action detection. In an offline setting, the best accuracy of subaction recognition is 56.3% [22]. This is, however not comparable to our results, since online action detection is a much harder problem, as argued earlier.

5.1.2 TVSeries Dataset

The TVSeries Dataset [7] consists of 16 hours of episodes of recent TV series. There are 30 action classes annotated (like 'going up stairway' or 'drive car'). The dataset has three splits, for training, validation and testing. This dataset contains a huge amount of unannotated background data. Most annotated actions are not dependent on each other, so the (annotated) long-term dependencies are limited. Here too, short-term dependencies exist between frames of the same action, though.

5.2. Setup

For the Noisy MNIST dataset, we used the model developed by LeCun *et al*. [25] and trained it on their clean MNIST data (static images). On the noisy test data we generated for these experiments, this model has an accuracy of 55%. The fc2 layer of this network is used as input for the LSTM and our feedback architecture. For the Breakfast dataset, we finetuned the two-stream network developed by Feichtenhofer *et al*. [11], that they first trained on the UCF101 dataset [32]. This model is a CNN that takes the appearance of the current frame as input for its first stream, and the optical flow over the last few frames for its second stream. This two-stream CNN has an accuracy of 22.7% on the Breakfast test data. We took the output of the fc6 layer as input for the LSTM and our feedback architecture. We also experimented with pose features on this dataset, that we extracted with the code provided by Cao *et al*. [5]. This way, we obtain the 2D coordinates of 18 keypoints. For the TVSeries Dataset, we start from the fc6 features provided by the authors, extracted from a frame-wise spatial VGG-net [29] finetuned on the dataset.

We experimentally discovered that an LSTM with 128 units works best. In our experiments, we train the LSTM with back-propagation. We unroll it in time over 250 frames to be able to recognize long-term dependencies. We use the Adadelta optimizer [37], because it does not need to set the learning rate. This way, our results are not affected by the tuning of this parameter. For evaluation, we use the accuracy over all frames for the Noisy MNIST and Breakfast Dataset, as suggested by [21]. For the TVSeries Dataset, we use the average precision and calibrated average precision proposed by the authors [7].

We compare with standard LSTM baselines: 1-layer LSTM with 128 units, and 2-layer LSTM with 128 units per layer.

	MNIST-Easy	MNIST-Hard
LSTM	92.34	89.26
2S-FN + Sum	92.73	90.48
2S-FN + Concatenation	92.64	91.10
2S-FN + Sum + extra layer	92.34	90.71
2S-FN + Concatenation + extra layer	92.01	90.61
2S-FN + Weighted sum	92.86	91.13
2S-FN + Weighted sum + extra layer	92.41	91.01

Table 1. Accuracy on the artificial Noisy MNIST dataset: LSTM and our two-stream feedback network (2S-FN) for different versions of the combination unit.

	fc6	Pose
LSTM	28.19	8.16
LSTM 2 layers	26.78	7.58
2S-FN + Sum	28.92	13.79
2S-FN + Concatenation	28.54	13.89
2S-FN + Sum + extra layer	31.93	13.83
2S-FN + Concatenation + extra layer	32.55	13.48
2S-FN + Weighted sum	29.8	12.89
2S-FN + Weighted sum + extra layer	31.06	12.83

Table 2. Accuracy on the first split of the Breakfast dataset: LSTM and our two-stream feedback network (2S-FN) for different versions of the combination unit. Inputs are the fc6 features of a two-stream CNN or pose features.

5.3. Results

We experiment with our two-stream feedback network of section 4 on the Noisy MNIST toy dataset. The results can be found in table 1. As discussed in section 4, we tried multiple variants for the combination unit. Most strategies are better than the baseline LSTM, especially on the MNIST-Hard dataset, where the temporal dependencies are less straightforward.

The results on the first split of the Breakfast dataset are shown in table 2. Here, the best models all have an extra LSTM layer. This makes sense: the extra LSTM layer is capable of making a decision based on information from the past. It is able to detect bursts of mistakes of one of the streams. Further, the concatenation seems to be a bit better than the sum, probably because the concatenation does not compress the output of the two streams. The 'concatenation + extra layer' model performs better than a two-layer

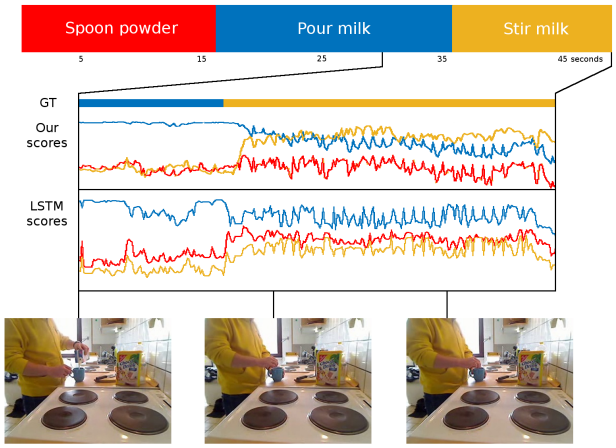

Figure 3. In this instance of 'making cocoa', the LSTM recognizes a spoon and its scores for both 'spoon powder' and 'stir milk' increase. Our method remembers having already seen 'spoon powder', so it knows this must be 'stir milk'.

LSTM, but it also has more parameters. The 'sum + extra layer', however, has only a few more parameters than the two-layer LSTM (due to the second stream with low input dimensions), yet it still performs significantly better.

The detection is significantly better when we use the CNN features as input as compared to the pose features. CNN features contain information on the whole image. On this dataset, especially the objects in the frame contain valuable information: whether the person manipulates a frying pan or a cup of milk, for instance. The pose features discard all this information, resulting in a worse detection. Note, however, that with 47 classes, the detection accuracy is still much higher than random.

When we use a fully-connected layer in the upper stream instead of an LSTM, the score of the 'concatenation + extra layer' model drops to 26.27%. It seems that the LSTM of the upper stream does a better job at compressing the information of the previous frames. It can filter out some noise. In theory, the LSTM layer after the concatenation unit should be able to do this filtering as well. In practice, however, it is clearly not up to the task.

Some examples where our method works better than the standard LSTM can be found in figures 3, 4 and 5. For further explanations, please look at the captions of the figures.

The results on the TVSeries Dataset can be found in tabel 3. On this dataset, we report the mean average precision and calibrated average precision, as suggested by the authors. Our baseline LSTM model is already significantly better than the LSTM used by the authors, even though we only use 128 LSTM units, as opposed to the 512 reported in [7]. This improvement is likely because we unroll the LSTM over more frames during training. De Geest *et al*. only unroll over 16 frames (less than a second), while we unroll over 250 frames (10 seconds). This way, the LSTM gets the opportunity to change its parameters to detect longer-term connections between actions.

	Input	mAP	cAP
LSTM	RGB only	3.8	71.6
LSTM 2 layers	RGB only	4.1	71.4
2S-FN + Sum	RGB only	4.3	72.3
2S-FN + Concatenation	RGB only	4.3	72.4
2S-FN + Sum + extra layer	RGB only	4.0	72.0
2S-FN + Concatenation + extra layer	RGB only	3.9	72.2
2S-FN + Weighted sum	RGB only	3.9	72.0
2S-FN + Weighted sum + extra layer	RGB only	3.9	71.1
CNN [7]	RGB only	1.9	60.8
LSTM [7]	RGB only	2.7	64.1
FV + SVM [7]	RGB + flow	5.2	74.3
RED-VGG [13]	RGB only	–	71.2
RED-TS [13]	RGB + flow	–	79.2

Table 3. Mean average precision and calibrated average precision on the TVSeries Dataset [7]; comparison with state of the art methods with similar RGB input and methods that also use optical flow.

On this dataset, we obtain our best detection scores with the 'concatenation' model. The extra layer after the combination unit is less effective. There are not many long-term temporal dependencies between the annotated actions of this dataset. The short-term dependencies are already captured by the temporal stream, so the task of the extra layer becomes easy enough to be absorbed by the fully-connected layer following it.

Gao *et al*. [13] report results on this dataset as well. They actually developed an action anticipation method, that tries to predict what actions will occur in the next frames and what the representation of these frames will be. As a side experiment, they test their strategy for online action detection by taking a very small anticipation time. Their RED-VGG model takes the 4096-dimensional VGG vector as input, similar to our methods. The RED-TS model uses features from a two-stream network (with appearance and optical flow) as input. Our results are better than theirs with the VGG features, even though they use significantly more parameters: 4096 LSTM units instead of our 128 units. The handcrafted FV + SVM approach still outperforms both, though. With two-stream CNN features, however, their detection scores are clearly superior: even better than the FV + SVM strategy. For our method as well, taking two-stream CNN features as input will likely boost the detection scores.

We tested our 'concatenation + extra layer' method on the Online Action Detection Dataset [26] as well, for reference. We used the 3D pose features provided by the authors. The baseline LSTM has a per-frame accuracy of 80.59%, while our method has only 68.69% accuracy. This dataset is explicitly collected to have no dependence between the actions. Moreover, it is quite small. Our model tries to find

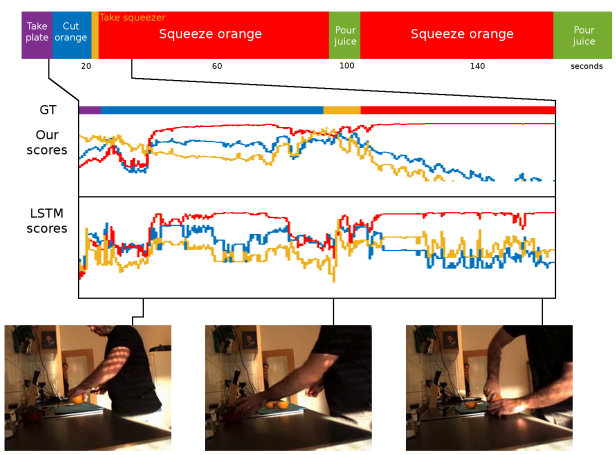

Figure 4. In this instance of 'making orange juice', both models increase the probability of 'squeeze orange' (wrong) and 'cut orange' (right) when the person grabs the orange. After the person has cut the orange, while he puts away his knife, our method already increases the probability of 'take squeezer'. The LSTM only realizes what is happening when the person has the squeezer in his hand. While the person is squeezing, the LSTM keeps the likelihood of other orange-related actions constant. Once our method is sure the person is squeezing, it rapidly decreases the probability that 'cut orange' or 'take squeezer' will occur again.

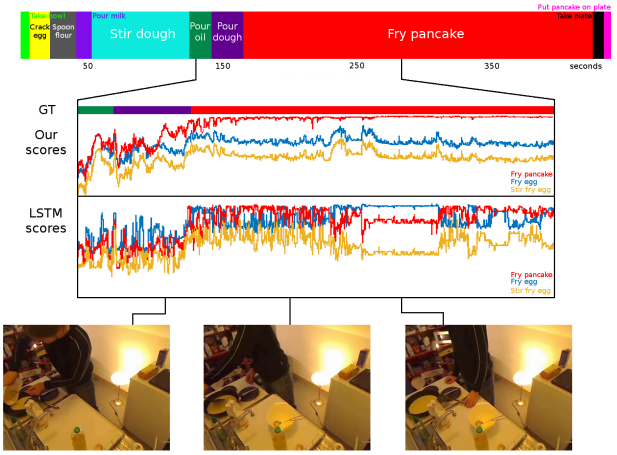

Figure 5. In this instance of 'making pancake', we compare the scores for 'fry pancake' with those for 'fry egg' and 'stir fry egg'. Our method correctly recognizes the person is frying pancakes, based on the earlier interaction with pancake ingredients. The score already increases while the person is pouring dough in the pan. The LSTM, on the other hand, is more confused. After a few seconds, it recognizes 'fry pancake', but later (when the person is only watching the pan), it changes its opinion to 'fry egg'. It keeps switching back and forth. Our method remembers the person is dealing with pancakes the whole time.

dependencies that are not there, and therefore overfits.

Online action recognition is most useful in a real-time setting. Our unoptimized code can process 1450 frames per second with a standard LSTM model and 1050 frames per second with our slowest model ('concatenation + extra layer'). The calculations are done on a GeForce GTX 750 Ti GPU. The extraction of appearance-based CNN features can be done at 55 frames per second. Real-time processing is therefore certainly possible.

6. Conclusion

LSTM seems a good choice for online action detection. It can model short-term and long-term patterns, and it receives its input sequentially and produces an output every step. In practice, however, the detection results are still quite low. We have designed a two-stream feedback network, where one stream focuses on the input interpretation and the other on the temporal patterns. Our experiments on synthetic and real-life data with a temporal structure show that a two-stream feedback network, where one stream focuses on the input interpretation and the other on the temporal patterns, helps for online action detection. This strategy is easy to implement and can be integrated with many others that build on top of an LSTM.

Acknowledgement

This work was supported by the KU Leuven GOA project CAMETRON.

References

[1] S. Baek, K. I. Kim, and T.-K. Kim. Real-time online action detection forests using spatio-temporal contexts. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 158–167. IEEE, 2017.

[2] F. Becattini, T. Uricchio, L. Ballan, L. Seidenari, and A. Del Bimbo. Am I done? Predicting action progress in videos. *arXiv preprint arXiv:1705.01781*, 2017.

[3] A. Bobick and J. Davis. Real-time recognition of activity using temporal templates. In *Applications of Computer Vision, 1996. WACV'96., Proceedings 3rd IEEE Workshop on*, pages 39–42. IEEE, 1996.

[4] L. Cao, Z. Liu, and T. S. Huang. Cross-dataset action detection. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 1998–2005. IEEE, 2010.

[5] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[6] Y. Cheng, Q. Fan, S. Pankanti, and A. Choudhary. Temporal sequence modeling for video event detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2227–2234, 2014.

[7] R. De Geest, E. Gavves, A. Ghodrati, Z. Li, C. Snoek, and T. Tuytelaars. Online action detection. In *European Conference on Computer Vision*, pages 269–284. Springer, 2016.

[8] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[9] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1110–1118, 2015.

[10] H. Fan, X. Chang, D. Cheng, Y. Yang, D. Xu, and A. G. Hauptmann. Complex event detection by identifying reliable shots from untrimmed videos. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 736–744. IEEE, 2017.

[11] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] A. Gaidon, Z. Harchaoui, and C. Schmid. Actom sequence models for efficient action detection. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3201–3208. IEEE, 2011.

[13] J. Gao, Z. Yang, and R. Nevatia. Red: Reinforced encoder-decoder networks for action anticipation. In *BMVC*, 2017.

[14] G. Garcia-Hernando and T.-K. Kim. Transition forests: Learning discriminative temporal transitions for action recognition and detection. CVPR, 2017.

[15] M. Hoai and F. De la Torre. Max-margin early event detectors. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2863–2870. IEEE, 2012.

[16] M. Hoai and F. De la Torre. Max-margin early event detectors. *International Journal of Computer Vision*, 107(2):191–202, 2014.

[17] D. Huang, S. Yao, Y. Wang, and F. De La Torre. Sequential max-margin event detectors. In *European conference on computer vision*, pages 410–424. Springer, 2014.

[18] M. Jain, J. Van Gemert, H. Jégou, P. Bouthemy, and C. G. Snoek. Action localization with tubelets from motion. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 740–747, 2014.

[19] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. In *International Conf. on Computer Vision (ICCV)*, pages 3192–3199, Dec. 2013.

[20] Y. Ke, R. Sukthankar, and M. Hebert. Efficient visual event detection using volumetric features. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 166–173. IEEE, 2005.

[21] H. Kuehne, A. B. Arslan, and T. Serre. The language of actions: Recovering the syntax and semantics of goal-directed human activities. In *Proceedings of Computer Vision and Pattern Recognition Conference (CVPR)*, 2014.

[22] H. Kuehne, J. Gall, and T. Serre. An end-to-end generative framework for video segmentation and recognition. In *Proc. IEEE Winter Applications of Computer Vision Conference (WACV 16)*, Lake Placid, Mar 2016.

[23] I. Laptev. On space-time interest points. *International journal of computer vision*, 64(2-3):107–123, 2005.

[24] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] Y. Li, C. Lan, J. Xing, W. Zeng, C. Yuan, and J. Liu. Online human action detection using joint classification-regression recurrent neural networks. In *European Conference on Computer Vision*, pages 203–220. Springer, 2016.

[27] P. Matikainen, M. Hebert, and R. Sukthankar. Representing pairwise spatial and temporal relations for action recognition. *Computer Vision–ECCV 2010*, pages 508–521, 2010.

[28] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[30] G. Singh, S. Saha, and F. Cuzzolin. Online real time multiple spatiotemporal action localisation and prediction. *arXiv preprint arXiv:1611.08563*, 2017.

[31] K. Soomro, H. Idrees, and M. Shah. Online localization and prediction of actions and interactions. *arXiv preprint arXiv:1612.01194*, 2016.

[32] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[33] S. Stein and S. J. McKenna. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2013), Zurich, Switzerland*. ACM, September 2013.

[34] K. Tang, L. Fei-Fei, and D. Koller. Learning latent temporal structure for complex event detection. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1250–1257. IEEE, 2012.

[35] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 3551–3558, 2013.

[36] S. Yeung, O. Russakovsky, G. Mori, and L. Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2678–2687, 2016.

[37] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.